Speculative Scheduling for Stochastic HPC Applications

Ana Gainaru, Guillaume Pallez, Hongyang Sun, Padma Raghavan

To cite this version:

Ana Gainaru, Guillaume Pallez, Hongyang Sun, Padma Raghavan. Speculative Scheduling for Stochastic HPC Applications. ICPP 2019 - 48th International Conference on Parallel Processing, Aug 2019, Kyoto, Japan. hal-02158598

HAL Id: hal-02158598
https://inria.hal.science/hal-02158598

Submitted on 18 Jun 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Speculative Scheduling for Stochastic HPC Applications

Ana Gainaru
Vanderbilt University
Nashville, TN, USA
ana.gainaru@vanderbilt.edu

Guillaume Pallez (Aupy)
Inria, Univ. Bordeaux
Talence, France
guillaume.pallez@inria.fr

Hongyang Sun
Vanderbilt University
Nashville, TN, USA
hongyang.sun@vanderbilt.edu

Padma Raghavan
Vanderbilt University
Nashville, TN, USA
padma.raghavan@vanderbilt.edu

ABSTRACT

New emerging fields are developing a growing number of large-scale applications with heterogeneous, dynamic and data-intensive requirements that put a high emphasis on productivity and thus are not tuned to run efficiently on today's high performance computing (HPC) systems. Some of these applications, such as neuroscience workloads and those that use adaptive numerical algorithms, develop modeling and simulation workflows with stochastic execution times and unpredictable resource requirements. When they are deployed on current HPC systems using existing resource management solutions, it can result in loss of efficiency for the users and decrease in effective system utilization for the platform providers.

In this paper, we consider the current HPC scheduling model and describe the challenge it poses for stochastic applications due to the strict requirement in its job deployment policies. To address the challenge, we present speculative scheduling techniques that adapt the resource requirements of a stochastic application on-the-fly, based on its past execution behavior instead of relying on estimates given by the user. We focus on improving the overall system utilization and application response time without disrupting the current HPC scheduling model or the application development process. Our solution can operate alongside existing HPC batch schedulers without interfering with their usage modes. We show that speculative scheduling can improve the system utilization and average application response time by 25-30% compared to the classical HPC approach.

KEYWORDS

Scheduling algorithm, HPC runtime, stochastic applications

ACM Reference Format:
Ana Gainaru, Guillaume Pallez (Aupy), Hongyang Sun, and Padma Raghavan. 2019. Speculative Scheduling for Stochastic HPC Applications. In *Proceedings of the 48th International Conference on Parallel Processing (ICPP'19).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ICPP'19, August 05–08, 2019, Kyoto, Japan
© 2019 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 MOTIVATION

A growing number of new scientific fields start to emerge that have heterogeneous, dynamic and data-intensive requirements for the computing and storage systems of today's HPC platforms. Some of these fields focus on productivity but not efficiency, so they produce codes that have not been optimized or tuned for the HPC hardware and runtime. As a result, applications from these fields tend to have stochastic execution times and unpredictable resource requirements. This is typically the case, for example, in neuroscience applications [10], but also in some classical HPC applications that use adaptive numerical algorithms [24]. In addition, recent years have seen a convergence of HPC/Big Data/Machine Learning models, which introduce a new set of applications that are stochastic by nature and are expected to run on large-scale platforms. All of these applications exhibit unique characteristics that differentiate them from the traditional scientific HPC applications.

Current HPC batch schedulers are *reservation-based*, which require the user to specify the resource requirement of an application upon submission. Accurate resource estimation is the key to high performance: while under-estimation can result in a job being killed before completion, over-estimation may lead to poor resource utilization. Most HPC systems prefer over-estimation and then use backfilling algorithms to fill up the wasted time with small jobs. On Intrepid, an HPC system from the Argonne National Laboratory between 2008 and 2014, the execution times of over 80% of the jobs submitted were over-estimated by an average of more than one hour. While scheduling algorithms continue to evolve, current systems are becoming more complex, offering applications heterogeneous nodes with non-uniform memory hierarchies, burst buffers, vectorization units and unified virtual memory. It is therefore becoming harder for users to accurately estimate the resource consumption even for known scientific applications. Today's systems heavily rely on backfilling algorithms to hide this inaccuracy in resource estimation. However, studies have shown that the performance of backfilling algorithms is also directly related to how well the execution times of the small jobs can be estimated [9]. Thus, when running stochastic applications with unpredictable resource needs, users are forced to find ad-hoc solutions to overcome the resource estimation requirement of current schedulers in order to utilize the HPC resources, and this may lead to sub-optimal performance. One such solution used by the neuroscience community is to request reservations equal to the maximum execution time of an application based on the last few runs. Our prior study [10] has shown that, using this solution, over 25% of the jobs are killed by the scheduler for needing more time than requested while the rest create wasted time in the system of more than 4 hours per day.

In this paper, we present *speculative scheduling* as an extension to the classical reservation-based HPC batch schedulers. Specifically, we use speculation to determine the resource requirements of a stochastic application based on its past execution behavior. This is achieved by speculatively overwriting the user requested times of the application during submissions (including the initial one and all subsequent resubmissions in case of failures). Since the performance of backfilling greatly influences the efficiency of an HPC system, we also investigate speculative backfilling by allowing stochastic jobs to be backfilled into smaller spaces than their requested times. These speculative techniques can operate with any existing HPC scheduler without disrupting its usage mode while achieving better system utilization and application response time.

The main contributions of this paper are summarized as follows:

- A speculative scheduling algorithm that computes the optimal sequence of requested times for a stochastic application based on a probability distribution of execution time from its past behavior as well as an incoming stream of small jobs that can be explored by the backfilling algorithm.
- A speculative backfilling algorithm that can either be integrated on its own into an HPC scheduler or work together with our speculative scheduler. The algorithm speculatively schedules stochastic jobs (based on a cost model) that do not fit in a given backfilling window based on its original requested time when there is no available fitting job.
- A runtime framework that combines speculative scheduling/backfilling with current HPC batch schedulers. We implement this framework into a simulator, demonstrating how speculative techniques can be naturally integrated into reservation-based scheduling schemes.
- A extensive set of simulations based on both synthetic workloads and realistic neuroscience applications. The results show that speculative scheduling can significantly benefit stochastic applications in terms of both system-level and user-level performance metrics under practical HPC scheduling scenarios.

The rest of this paper is organized as follows. Section 2 presents the related work, highlighting an overview of the HPC schedulers and some stochastic scheduling techniques. Section 3 describes the models and metrics we use for speculative scheduling. Details and analyses for the proposed speculative algorithms, either for overwriting the requested times at submission or at backfilling times, are presented in Section 4. Section 5 presents the simulation results for speculative scheduling and backfilling. This section also includes a simulation based on the characteristics of real neuroscience applications combined with properties of HPC workloads on Intrepid, a large-scale system from the Argonne National Laboratory. Finally, Section 6 provides brief concluding remarks.

2 RELATED WORK

In this section, we briefly review some related work on the current scheduling practice in HPC systems, as well as some prior efforts on coping with stochastic applications.

HPC Schedulers. Reservation-based batch scheduling with priority queues and backfilling represents the de facto approach in HPC. It is adopted by many commonly used resource managers, such as Slurm [27], Torque [18] and Moab [5]. Most of these scheduling implementations use an iteration algorithm in order to decide which jobs to run and on what resources. Due to the use of reservation, there exists a trade-off between the system efficiency and application performance, and different HPC schedulers employ different priority schemes and backfilling algorithms to deal with this trade-off while ensuring no job starvation. These reservation-based schedulers also rely on the users to provide reasonably accurate runtime for each submitted job. While this works well for applications with deterministic resource needs, for stochastic jobs with large variations in the walltime, it will lead to either over-estimation or under-estimation of the resources, thus degrading system utilization and/or application response time [10]. Clusters of commodity servers that use computing frameworks, such as MapReduce [6] or Dryad [15], offer alternative solutions to running HPC workloads. Schedulers for these frameworks, such as YARN [22] and Mesos [13], offer several distinct features to the management of the workloads (such as fairness, resource negotiation), but they generally also require accurate information regarding the applications' resource demands.

Stochastic Scheduling. Many prior works have considered stochastic job scheduling under various models and assumptions (see, e.g., [3, 4, 7, 16, 20, 21, 25]). We refer interested readers to the book by Pinedo [17] for a comprehensive survey of different stochastic scheduling problems, and to the book chapter [11] for a comparison of different stochastic task-resource systems. Most of these works, however, do not consider the problem under the reservation-based scheduling context. In our prior works [1, 10], we have proposed near optimal reservation strategies for a single job without backfilling, and extended it to a set of sequential jobs in a reservation-based HPC scheduling environment.

In this paper, we study a more realistic scenario in HPC with parallel stochastic jobs and backfilling considerations. Our speculative solutions are not intended to replace the existing batch schedulers, but to augment them with the possibility to speculate the best resource requirements and scheduling options based on the jobs' past stochastic behaviors.

3 MODELS AND PERFORMANCE METRICS

In this section, we describe the models and assumptions we make for scheduling stochastic jobs. We also present the metrics for evaluating a scheduling algorithm in terms of both system-level and user-level performance.

3.1 Job Models

We consider scheduling a batch $\mathcal{J} = \{J_1, J_2, \ldots, J_M\}$ of *large* stochastic jobs along with a stream $\mathcal{B}$ of *small* jobs (for backfilling) on a system with P identical processors.

Each stochastic job $J_j \in \mathcal{J}$ has a specified processor allocation p_j (i.e., jobs are assumed to be rigid). However, the execution time of the job is unknown, and is assumed to be randomly and uniformly sampled from a given probability distribution law $\mathcal{D}_j$, whose density function (PDF) is f_j and cumulative distribution function (CDF) is F_j. Note that although a job is stochastic, the execution time of a particular instance of the job is deterministic, so two consecutive executions of the same instance will have the same duration. The probability distribution for each job J_j is assumed to be nonnegative, since it models execution times, and is defined on a finite support $[a_j, b_j]$, where $0 \leq a_j < b_j$. Hence, the execution time of job J_j is a random variable X_j with $\mathbb{P}(X_j \leq T) = F_j(T) = \int_{a_j}^{T} f_j(t)dt$. Our experience with neuroscience workloads [10] shows that this model is consistent with large-scale stochastic applications.

The stream of small jobs arrives in the system and needs to be scheduled using backfilling along with the large stochastic jobs. In practice, small jobs are typically sequential ones with an arrival rate λ and an average execution time ε much smaller than that of the large jobs. This can be understood as every $1/\lambda$ units of time, approximately ε additional work is added to the execution queue. To model these small jobs, we make a continuous approximation: a stream of work arrives continuously in the queue with a rate $Z = \lambda\varepsilon$. Assuming that all processors have unit speed, the work rate needs to satisfy $Z \leq P$ in order not to overflow the queue. Moreover, this work is fully parallelizable and can be dynamically scheduled on an arbitrary number of processors. Note that, in the evaluation (Section 5), we study the impact of this continuous approximation by simulating discrete small jobs.

3.2 Scheduling Models

Speculative Scheduling for Large Stochastic Jobs. To schedule the large stochastic jobs, we consider speculative scheduling using the reservation-based model, which is widely adopted by HPC batch schedulers. For each job $J_j \in \mathcal{J}$, the scheduler makes a reservation of p_j processors with initial duration, say $t_{j,1}$, on the platform at time $s_{j,1}$. The job then runs starting from time $s_{j,1}$ until either it has successfully completed or the reservation time has elapsed, whichever comes first. In the latter case, the job does not complete successfully and needs to be rescheduled with a longer duration, say $t_{j,2} > t_{j,1}$, along with a new starting time $s_{j,2}$. If the job still does not complete successfully, then a third reservation $(t_{j,3}, s_{j,3})$ needs to be made and so on until the job eventually completes successfully. Hence, for each large job J_j, the scheduler needs to make a sequence of speculative reservations:

$$S_j = \langle (t_{j,1}, s_{j,1}), (t_{j,2}, s_{j,2}), \ldots, (t_{j,i}, s_{j,i}), \ldots \rangle$$

where $t_{j,i+1} > t_{j,i}$ and $s_{j,i+1} \geq s_{j,i} + t_{j,i}$ for all $i \geq 1$.

Backfilling for Small Jobs. Unlike large jobs, no reservation is made for small jobs. Instead, they are scheduled by backfilling the work stream into the gaps created by the reservations of the large jobs or on-the-fly while no large job is running (e.g., when all large jobs have completed).

Scheduling Objective. The objective is to design scheduling strategies, i.e., a sequence S_j of reservations for each large job $J_j \in \mathcal{J}$ and execution time for the stream $\mathcal{B}$ of small jobs, that optimizes both system-level and user-level performance as described in Section 3.3. In addition, a scheduling strategy needs to respect the processor constraint: at any time, the sum of processors used by all running jobs should not exceed the total number P of available processors on the platform.

3.3 Performance Metrics

We consider two metrics to evaluate the performance of a scheduling strategy from both the user level and the system level.

We first define some notations. Given a schedule S for $\mathcal{J}$ and $\mathcal{B}$, let $\tau_S(X_j)$ be a random variable that denotes the successful completion time of the large job $J_j \in \mathcal{J}$, i.e., $\tau_S(X_j) = s_{j,i} + t_{j,i}$ for $t_{j,i-1} < X_j \leq t_{j,i}$.[1] We further define $\tau_S(\mathcal{J}) = \max_{j=1\ldots M} \tau_S(X_j)$ to be the completion time of the last successfully completed large job. For the small jobs, let $C_S(t)$ denote a random variable that indicates the amount of small work completed at time t, and $\sigma_S(\mathcal{J}, \mathcal{B}) = \inf\{t | t \geq \tau_S(\mathcal{J}), C_S(t) \geq \lambda\varepsilon t\}$ another random variable that indicates the first time the small work queue becomes empty after all the large jobs have completed.

We use the following performance metrics in this paper.

- *Expected system utilization:* This is a system-level metric that measures the expected utilization of the system when all the work has been completed, i.e.,

$$U_S = \mathbb{E}\left(\frac{\sum_{j=1}^{M} X_j + \lambda\varepsilon \cdot \sigma_S(\mathcal{J}, \mathcal{B})}{P \cdot \sigma_S(\mathcal{J}, \mathcal{B})}\right) \tag{1}$$

  Note that a failed reservation for a job is considered wasted and is therefore not counted towards the utilization.
- *Expected average job response time:* This is a user-level metric that measures the average response time for all the large jobs in expectation, i.e.,

$$C_S = \mathbb{E}\left(\frac{\sum_{j=1}^{M} \tau_S(X_j)}{M}\right) \tag{2}$$

4 SPECULATIVE SCHEDULING STRATEGIES

In this section, we present speculative scheduling strategies for large stochastic jobs in the presence of small backfilling jobs. To that end, we consider a multi-pronged approach, which first finds an optimal reservation sequence for each stochastic job independently of the other jobs in the system (Sections 4.1 and 4.2), and then uses a round-based greedy heuristic that uses the sequences to schedule all jobs and that relies on speculative backfilling (Section 4.3).

4.1 Optimal Reservation Sequence for a Single Stochastic Job

We first present a strategy for determining the optimal reservation sequence of a single stochastic job. The goal is to minimize the expected cumulative execution time (or makespan) of the job (until success) in the presence of the backfilling work stream. When only one large job exists in the system, this is equivalent to the optimal expected job response time and well approximates the optimal expected system utilization.

The optimal strategy in this section is derived for a particular type of execution time distribution: discrete distributions where the execution time values are equally spaced. The motivation behind this is: (i) we can easily approximate a continuous distribution with such a distribution (Section 4.2); (ii) in this context, we can provide an optimal algorithm for the problem with backfilling (Theorem 2).

[1]Here, we make the pessimistic assumption that the job completes at the end of the reservation period, since in the worst case there will be no small jobs to fill in the gap created due to over-estimation, thus resulting in waste of resources.

Definition 1 (Equally-Spaced Discrete Distribution (ESDD))**.** A discrete distribution $X \sim (v_i, f_i = \mathbb{P}(X = v_i))_{0 \le i \le n}$ is equally-spaced if there exists a $\delta = \frac{v_n - v_0}{n}$ such that $\forall i \le n$, we have:

$$v_i = v_0 + i \cdot \delta$$

In the following, we will consider a stochastic job with discrete execution time $X \sim \text{ESDD}(v_0, \delta, n, (f_i)_i)$, and design a strategy that minimizes the expected makespan of the single job.

Note that the possibility to backfill with small jobs in the system encourages strategies that slightly over-estimate a reservation time: Indeed, in case the job is shorter, this over-estimation can be compensated by the backfilling jobs. Recall that, in Section 3, we have made the assumption that the arrival of the backfilling jobs is independent of the status of the system (whether a large job is running or not) and can be modeled by a stream of work with rate Z. Here, for fairness, we assume that the stochastic job is *responsible* for a fraction of the small jobs that is proportional to its processor allocation. Hence, suppose the large job is allocated p processors, then the rate of work that arrives in the queue of the large job is given by $\zeta = Z \cdot \frac{p}{P} = \lambda\varepsilon \cdot \frac{p}{P} \le p$.

We start by providing an equation for the expected makespan of the stochastic job in the presence of backfilling, which we will use later for the design of an optimal solution.

Theorem 1. *Consider a stochastic job with $X \sim ESDD(v_0, \delta, n, (f_i)_i)$, a sequence $S = (v_{\pi(1)}, \dots, v_{\pi(|S|)} = v_n)$ of reservations, and a backfilling rate ζ. Let T be a random variable that represents the makespan of executing X. We have:*

$$\mathbb{E}(T) = \sum_{m=1}^{|S|} \left(v_{\pi(m)} \cdot \Big(\sum_{\ell=\pi(m)+1}^{\pi(|S|)} f_\ell \Big) + \sum_{\ell=\pi(m)+1}^{x_m^S} v_{\pi(m+1)} \cdot f_\ell \right.$$
$$\left. + \frac{1}{1-\zeta} \sum_{\ell=x_m^S+1}^{\pi(m+1)} v_\ell f_\ell + \frac{\zeta}{1-\zeta} \sum_{i=1}^{m} v_{\pi(i)} \cdot \sum_{\ell=x_m^S+1}^{\pi(m+1)} f_\ell \right) \quad (3)$$

where x_m^S is the unique integer such that:

$$v_{x_m^S} \le \min\left(v_{\pi(m)},\ (1-\zeta) v_{\pi(m+1)} - \zeta \sum_{i=1}^{m} v_{\pi(i)} \right) < v_{x_m^S+1}$$

Proof. We consider the case such that $v_{\pi(m)} < X \le v_{\pi(m+1)}$, and evaluate the backfilling work that is accumulated over the execution of X: (1) During the actual execution of X (total time of $T_1 = \sum_{i=1}^m v_{\pi(i)} + X$), we accumulate $T_2 = \zeta T_1$ units of backfilling work; (2) Then, during the execution of T_2, we accumulate ζT_2 units of backfilling work, etc. After the job is successfully executed, the total amount of backfilling work is $\sum_{x=1}^{\infty} \zeta^x T_1 = \frac{\zeta}{1-\zeta} T_1$. Finally, the total work to execute is $\frac{\zeta}{1-\zeta} T_1 + T_1 = \frac{1}{1-\zeta} T_1$. Hence, the random variable T is given by:

$$T = \max\left(\sum_{i=1}^{m+1} v_{\pi(i)}, \frac{1}{1-\zeta} \Big(\sum_{i=1}^{m} v_{\pi(i)} + X \Big) \right)$$

The first term above is when the remaining time at the end of the last reservation (i.e., $v_{\pi(m+1)}$) is enough to execute all the backfilling work accumulated, while the second term is when it is not. We can further define:

$$X_{m+1}^S = \min\left((1-\zeta) v_{\pi(m+1)} - \zeta \sum_{i=1}^{m} v_{\pi(i)},\ v_{\pi(m)} \right) \quad (4)$$

to be the *threshold* execution time for the job such that the makespan can be expressed as:

$$T = \begin{cases} \sum_{i=1}^{m+1} v_{\pi(i)} & \text{if } v_{\pi(m)} < X \le X_{m+1}^S \\ \frac{1}{1-\zeta} \left(\sum_{i=1}^{m} v_{\pi(i)} + X \right) & \text{if } X_{m+1}^S < X \le v_{\pi(m+1)} \end{cases} \quad (5)$$

Let us denote by x_m^S the unique integer such that $v_{x_m^S} \le X_{m+1}^S < v_{x_m^S+1}$. With Eq. (5), we can write the expected makespan with backfilling as:

$$\mathbb{E}(T) = \sum_{m=1}^{|S|} \left(\sum_{\ell=\pi(m)+1}^{x_m^S} \Big(\sum_{i=1}^{m+1} v_{\pi(i)} \Big) \cdot f_\ell \right.$$
$$\left. + \sum_{\ell=x_m^S+1}^{\pi(m+1)} \frac{1}{1-\zeta} \Big(\sum_{i=1}^{m} v_{\pi(i)} + v_\ell \Big) \cdot f_\ell \right)$$
$$= \sum_{m=1}^{|S|} \left(\sum_{\ell=\pi(m)+1}^{x_m^S} \Big(\sum_{i=1}^{m} v_{\pi(i)} \cdot f_\ell + v_{\pi(m+1)} \cdot f_\ell \Big) \right.$$
$$\left. + \sum_{\ell=x_m^S+1}^{\pi(m+1)} \Big(\frac{v_\ell f_\ell}{1-\zeta} + \sum_{i=1}^{m} v_{\pi(i)} \cdot f_\ell + \frac{\zeta}{1-\zeta} \sum_{i=1}^{m} v_{\pi(i)} \cdot f_\ell \Big) \right)$$

Finally, using the following equation:

$$\sum_{m=1}^{|S|} \left(\sum_{\ell=\pi(m)+1}^{x_m^S} \Big(\sum_{i=1}^{m} v_{\pi(i)} \cdot f_\ell \Big) + \sum_{\ell=x_m^S+1}^{\pi(m)+1} \Big(\sum_{i=1}^{m} v_{\pi(i)} \cdot f_\ell \Big) \right)$$
$$= \sum_{m=1}^{|S|} \left(\sum_{\ell=\pi(m)+1}^{\pi(m+1)} \Big(\sum_{i=1}^{m} v_{\pi(i)} \cdot f_\ell \Big) \right) = \sum_{m=1}^{|S|} \left(\sum_{i=1}^{m} v_{\pi(i)} \cdot \Big(\sum_{\ell=\pi(m)+1}^{\pi(m+1)} f_\ell \Big) \right)$$
$$= \sum_{i=1}^{|S|} \left(v_{\pi(i)} \cdot \Big(\sum_{\ell=\pi(i)+1}^{\pi(|S|)} f_\ell \Big) \right)$$

we can obtain the result as shown in Eq. (3). □

Note that, when ζ approaches 0, X_{m+1}^S is very close to $v_{\pi(m+1)}$ and the objective reduces to the case without backfilling [1]. On the other hand, when ζ is large, X_{m+1}^S gets close to $v_{\pi(m)}$ and the objective function is reduced to $\sum_i \left(v_{\pi(i)} \cdot \sum_{\ell=\pi(i)+1}^{n} f_\ell \right)$, which is minimized when the strategy chooses a single reservation v_n.

Generally, we can construct a dynamic programming algorithm to compute the optimal reservation sequence for the stochastic job with backfilling.

Theorem 2. *Consider a stochastic job with $X \sim ESDD(v_0, \delta, n, (f_i)_i)$ and a backfilling rate ζ. The minimal expected makespan of the job is returned by $E(0, 0, 0)$, where:*

$$E(i, m, k) = \min_{i < i' \le n} \left(v_{i'} \cdot \sum_{\ell=i+1}^{x(i,m,k,i')} f_\ell + \frac{\zeta(m v_0 + k\delta)}{1-\zeta} \sum_{\ell=x(i,m,k,i')+1}^{i'} f_\ell \right.$$
$$\left. + \frac{1}{1-\zeta} \sum_{\ell=x(i,m,k,i')+1}^{i'} f_\ell v_\ell + v_{i'} \sum_{\ell=i'+1}^{n} f_\ell + E(i', m+1, k+i') \right) \quad (6)$$

with $x(i, m, k, i') = \max\left(i, \left\lfloor i' - \zeta(i'+k) - \zeta\frac{(m+1)v_0}{\delta}\right\rfloor\right)$ *and for all* m, k, *we have* $E(n, m, k) = 0$. *The complexity of the algorithm is* $O(n^5)$.

Proof. We define $E(i, m, k)$ to be the expected execution time of the job when: (1) There are m reservations $(v_{\pi(1)}, v_{\pi(2)}, \dots, v_{\pi(m)})$ that are smaller than or equal to v_i; (2) $\sum_{\ell=1}^{m} v_{\pi(\ell)} = m \cdot v_0 + k \cdot \delta$; and (3) $v_i = v_{\pi(m)}$ is the m-th reservation. Given these constraints, we look for i', the index of the next reservation, that minimizes the expected makespan.

Note that $x(i, m, k, i') = \max\left(i, \left\lfloor i' - \zeta(i'+k) - \zeta\frac{(m+1)v_0}{\delta}\right\rfloor\right)$ is the index of the threshold value as defined in Eq. (4):

$$\begin{aligned} X^S_{m+1} &= (1-\zeta)v_{i'} - \zeta \sum_{\ell=1}^{m} v_{\pi(\ell)} \\ &= v_0 + \delta\left(i' - \zeta(i'+k) - \zeta\frac{(m+1)v_0}{\delta}\right) \end{aligned}$$

We define the subequation of Eq. (3) as:

$$\begin{aligned} E_{\pi(i)} &= \sum_{m=i}^{|S|}\left(v_{\pi(m)} \cdot \Big(\sum_{\ell=\pi(m)+1}^{\pi(|S|)} f_\ell\Big) + v_{\pi(m+1)} \cdot \sum_{\ell=\pi(m)+1}^{x^S_m} f_\ell \right. \\ &\quad \left. + \frac{1}{1-\zeta}\sum_{\ell=x^S_m+1}^{\pi(m+1)} v_\ell f_\ell + \frac{\zeta}{1-\zeta}\sum_{j=1}^{m} v_{\pi(j)} \cdot \sum_{\ell=x^S_m+1}^{\pi(m+1)} f_\ell\right) \\ &= E_{\pi(i+1)} + \left(v_{\pi(i)} \cdot \Big(\sum_{\ell=\pi(i)+1}^{n} f_\ell\Big) + v_{\pi(i+1)} \cdot \sum_{\ell=\pi(i)+1}^{x^S_i} f_\ell \right. \\ &\quad \left. + \frac{1}{1-\zeta}\sum_{\ell=x^S_i+1}^{\pi(i+1)} v_\ell f_\ell + \frac{\zeta}{1-\zeta}\sum_{j=1}^{i} v_{\pi(j)} \cdot \sum_{\ell=x^S_i+1}^{\pi(i+1)} f_\ell\right) \end{aligned} \tag{7}$$

In addition, we have (by construction):

$$x^S_m = x\left(\pi(m), m, \frac{\sum_{\ell=1}^{m}(v_{\pi(\ell)} - v_0)}{\delta}, \pi(m+1)\right).$$

We can now express $E(i, m, k)$ using the following dynamic programming formulation:

$$\begin{aligned} E(i, m, k) &= \min_{i < i' \le n}\left(v_{i'} \cdot \sum_{\ell=i+1}^{x(i,m,k,i')} f_\ell + \frac{\zeta(mv_0 + k\delta)}{1-\zeta}\sum_{\ell=x(i,m,k,i')+1}^{i'} f_\ell \right. \\ &\quad \left. + \frac{1}{1-\zeta}\sum_{\ell=x(i,m,k,i')+1}^{i'} f_\ell v_\ell + v_{i'}\sum_{\ell=i'+1}^{n} f_\ell + E(i', m+1, k+i')\right) \end{aligned}$$

which we initialize with for all $m \in \{1, \cdots, n\}$ and $k \in \{n + m(m-1)/2, \cdots, m(2n-m+1)/2\}$ (the smallest k is when the reservations are $v_1, v_2, \cdots, v_{m-1}, v_n$, and the largest is when the reservations are $v_{n-m+1}, v_{n-m+2}, \cdots, v_{n-1}, v_n$), and $E(n, m, k) = 0$.

We can show by decreasing induction that, for all m, we have $E_{\pi(m)} \ge E(\sigma(m), m, k_m)$, where $k_m = \frac{\sum_{i=1}^{m}(v_{\pi(i)} - v_0)}{\delta}$. This is true for $\pi(|S|) = n$, i.e., $E_{\pi(|S|)} = 0 = E(n, m, k_m)$. Then for each m, we can replace the value of $E_{\pi(i+1)}$ in Eq. (7) and minimize for $E(\sigma(m), m, k_m)$. This proves the optimality of $E(0, 0, 0)$, which is smaller than the expected makespan of any reservation strategy.

Since there are $O(n^4)$ entries in the dynamic programming table, and computing each entry relies on at most n other entries, the complexity of the algorithm is $O(n^5)$. □

We point out that although the complexity of the above dynamic programming algorithm is seemingly high, the optimal solution typically has just a few reservations ($m \le 5$), which limits the complexity in m and k so that the total complexity is $O(n^3)$. Further, When used in conjunction with the discretization scheme of a continuous distribution (Section 4.2), it converges quite fast (with $n < 30$) for the distributions we studied. Hence, the algorithm can be used in practice to efficiently approximate the optimal reservation sequence of a continuous distribution.

For the remainder of the paper, we will refer to the above algorithm as *Time Optimal* (or *TOptimal*) when we use $E(0, 0, 0)$ and $\zeta = 0$ to compute the reservation sequence, which is called the TOptimal sequence. For $0 < \zeta < 1$, the algorithm adapts to an incoming stream of backfilling jobs, thus the algorithm and the resulting reservation sequence are referred to as *Adaptive Time Optimal* (or *ATOptimal*).

4.2 Equally-Spaced Discretization Scheme

The result of the previous section are optimal for a specific type of probability distribution: Equally-Spaced Discrete Distributions (ESDD). In order to use it in a more general framework, we apply a discretization scheme for a continuous distribution in this section.

Given a continuous distribution with finite support $[a, b]$, where $0 \le a < b$, we can construct a discrete distribution as follows. By choosing the number n of discrete values we will sample from the continuous distribution, we get a set of $n + 1$ pairs $(v_i, f_i)_{i=0\dots n}$, where the v_i's represent the discrete execution times of the jobs, and the f_i's represent the corresponding probabilities. The discretization scheme makes the discrete execution times equally spaced in the interval $[a, b]$. Thus, for all $i = 1, 2, \dots, n$, we have:

$$v_i = a + i \cdot \delta \text{ and } f_i = F(v_i) - F(v_{i-1})$$

where $\delta = \frac{b-a}{n}$ is the space between two consecutive execution time values and $(v_0, f_0) = (a, F(a))$ by definition.

4.3 Scheduling for a Set of Stochastic Jobs

We now describe scheduling strategies for a set $\mathcal{J} = \{J_1, J_2, \dots, J_M\}$ of stochastic jobs. The problem is clearly NP-complete because it contains the problem with deterministic jobs as a special case. To solve the problem, we first compute a reservation sequence S_j independently for each job $J_j \in \mathcal{J}$ as described in Sections 4.1 and 4.2. We then use a round-based greedy scheduling heuristic with speculative backfilling to schedule the jobs.

Round-Based Greedy Scheduling. To schedule a set of jobs, we employ a greedy approach that works in *rounds*: In the first round, each job J_j takes its first reservation $t_{j,1}$, and the jobs are prioritized in the non-increasing order of *processor-time product*, defined as $p_j \cdot t_{j,1}$. The jobs are then scheduled greedily in this order at the earliest time possible in the system. Once the first round completes, the jobs that fail due to insufficient first reservation are then scheduled in the second round using their second reservations. This process repeats until all jobs eventually complete. During any round, the

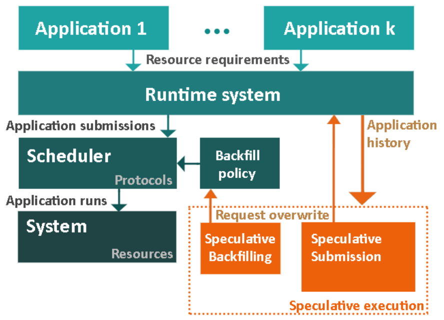


Figure 1: The workflow of our simulator. Teal-colored modules are used for classical HPC batch scheduler, and orange-colored modules are used for speculative scheduling.

stream of small jobs is used to backfill any gap created by the stochastic jobs due to: (1) over-estimation of the jobs' reservation times; and (2) imperfect packing of the jobs in the schedule.

Speculative Backfilling. To improve the system utilization and job response time, we also attempt to use backfilling to schedule the failed jobs in each round. Suppose a job J_j failed in a round (due to insufficient reservation $t_{j,k}$) but can fit in a gap later in the round with its next reservation $t_{j,k+1}$, we will then schedule the job there instead of delaying it to the next round. Moreover, since the jobs are stochastic, we can *speculatively* backfill a job even if its reservation is larger than the length of the gap, with the hope that the job may complete in this gap with some probability. When more than one such jobs are available, we choose the one that maximizes the expected utilization of the gap. Specifically, for a gap with q processors and duration d, and a subset $\mathcal{J}'$ of stochastic jobs for potential backfilling, we choose a job J_j that maximizes the expected utilization of the gap as follows:

$$\max_{J_j \in \mathcal{J}'} G_j = \frac{p_j \int_{a'_j}^{d} t \cdot f'_j(t)dt}{q \cdot d}$$

where $p_j \leq q$ is the processor allocation of the job, a'_j is the updated lower bound on the execution time of the job (possibly after previously failed reservations), and $f'_j(t) = f_j(t|t \geq a'_j)$ is the updated PDF of the job.

5 EXPERIMENTS

In this section, we use simulation to evaluate the performance of our speculative scheduling approaches. We conduct experiments using a simulator for HPC schedulers [8], which works on an input of applications with given resource requirements and submission times. The diagrams in teal shades from Figure 1 show the original workflow of the simulator. This workflow has been used and validated in [10] for analyzing the characteristics and performance of neuroscience applications.

We have modified the simulator by adding speculative components in the scheduling process. Specifically, the runtime system adjusts each application's submission by replacing its requested time with the value computed by the algorithm presented in the previous section. In addition, the runtime system is responsible for resubmitting any failed job with a corresponding new walltime request if necessary. For example, when using the TOptimal algorithm, if an application J_j is submitted by the user with a requested time t, this value is overwritten by the runtime to be $t_{j,1}$, corresponding to the first value in the reservation sequence given by TOptimal. If the application fails, the runtime system resubmits J_j requesting $t_{j,2}$ time and so on.

We have also configured the simulator to work with speculative backfilling by temporarily overwriting the requested time of an application just for the additional speculative run. Applications that fit in a given backfilling gap will be chosen before speculative ones. If there is still space left after all applications have been considered, we use the algorithm in Section 4.3 to make a speculative run that maximizes the expected utilization of the gap. If the application fails after the speculative run, we simply reset its normal requested time and add it back to the top of the waiting queue.

5.1 Simulated Scenarios

We are interested in simulating the following scenarios:

- *Scenario 1*: A comparison of classical HPC scheduler and speculative scheduler (TOptimal) that uses the sequence of requested times at submission time;
- *Scenario 2*: A comparison of classical HPC scheduler and speculative scheduler (TOptimal) with and without speculative backfilling;
- *Scenario 3*: A comparison of classical HPC scheduler and speculative scheduler (ATOptimal) when the system has an incoming stream of small jobs for backfilling;
- *Scenario 4*: A simulation of two weeks of execution on a large-scale machine with stochastic neuroscience applications.

The requested time for an application in the classical HPC scheduler is usually chosen to minimize the number of failures the application might encounter. For stochastic and unpredictable applications, this usually corresponds to the highest execution time. In practice, users can either submit with the highest execution time from the previous runs for a given application (which in our case corresponds to the distribution's upper bound) or use a more complex approach. The neuroscience applications from Vanderbilt are extremely dynamic, so their users choose to adapt the requested time to each new module development and algorithmic update [12]. For this reason, they typically use the last few runs of a given application (instead of the entire history) for choosing the highest value to be used for the next submission. In our experiments, we compare speculative scheduling with both approaches (choosing the distribution upper bound will be called classical HPC while using the adaptive method will be called Neuroscience).

5.2 Simulation Setup

We configure the applications to have four different walltime distributions: (1) Truncated Normal with $\mu = 8$ and $\sigma = 2$, and values from 6 to 16 hours; (2) Beta with $\alpha = 2$ and $\beta = 2$, and values from a few seconds to one hour; (3) Exponential with $\lambda = 1$ and values between a few seconds to 16 hours; and (4) Bounded Pareto with values from one hour to 20 hours and $\alpha = 2.1$.

Table 1: Reservation sequences for a Truncated Normal distribution with values from $a = 0$ to $b = 20$ hours, and $\mu = 8$, $\sigma = 2$ for different algorithms and scenarios.

Algorithm	Sequence of requests (in hours)
Classical HPC	20.0
TOptimal	10.8, 13.4, 15.4, 17.1, 18.7, 20.0
ATOptimal ($\zeta = 0.1$)	10.86, 13.91, 18.69, 20.0
ATOptimal ($\zeta = 0.5$)	13.04, 20.0
ATOptimal ($\zeta = 0.9$)	17.39, 20.0

We simulate a machine with $P = 100$ processors, and the processor allocations of the jobs are generated according to four cases: (1) Each job requests the entire machine for execution (Full); (2) Each job requests half of the machine for execution (Half); (3) Jobs are allocated with different numbers of processors in $[1, P]$ following a Truncated Normal distribution with $\mu = 0.5P$ and $\sigma = 0.3P$ (Truncnormal); and (4) Jobs are allocated with different numbers of processors following a Beta distribution with $\alpha = 2, \beta = 2$ (Beta). In this last case, since the Beta distribution produces values in [0, 1], the resulting processor allocations are scaled to be in $[1, P]$.

In the experiments, we generate a set of applications (with their processor allocations and execution time distributions) that will be used by all algorithms. After the set is generated, we compute the reservation sequence for each algorithm we want to analyze. As an example, Table 1 shows the reservation sequences when the application follows a Truncated Normal distribution with execution times ranging from a few seconds to 20 hours, and an average of 8 hours and a standard deviation of 2 hours. The classical HPC approach will always ask for 20 hours (the upper bound), while our TOptimal algorithm will generate a sequence of requests (initially asking for 10.8 hours, and if the application failed, resubmission will ask for 13.4 hours, etc.). The ATOptimal strategy tends to ask for larger times than TOptimal as the stream of small jobs can be used to backfill the gaps due to over-estimation.

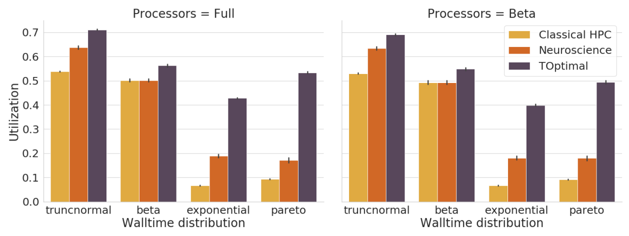
In the experiments, we start by simulating the simplest scenario that uses applications requiring the same number of processors and whose execution times follow the same distribution. With each experiment we relax restrictions until Section 5.6 where we simulate the behavior of real applications.

All experiments use $M = 100$ large jobs, unless otherwise specified. In each simulation, we make 50 runs and present the average results. When small jobs are included, their walltimes follow the same distributions as the ones for large jobs but scaled down by a factor of 100. Their average walltime is used to derive the rate Z for the work stream used by the ATOptimal algorithm.

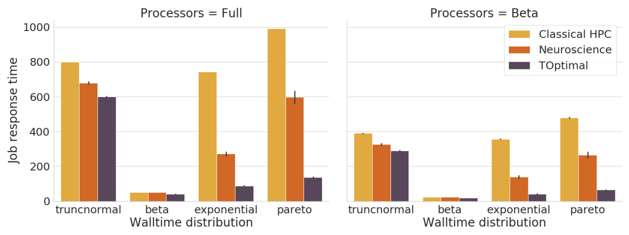
5.3 Speculative Scheduling (Scenario 1)

In this section, we compare speculative scheduling with the classical HPC and Neuroscience approaches. Figure 2

The percentage improvements are similar for different parameters of each distribution, so we chose the parameters that reflect closely our observations of realistic stochastic applications. Overall, speculative scheduling gives an average of 10-11% improvement in system utilization for the Truncated Normal and Beta distributions, and a 12-14% improvement in average job response time compared



(a) System utilization



(b) Average job response time

Figure 2: System utilization and average job response time under different walltime distributions for jobs that occupy the entire machine (Full) and jobs whose processor allocations follow the Beta distribution.

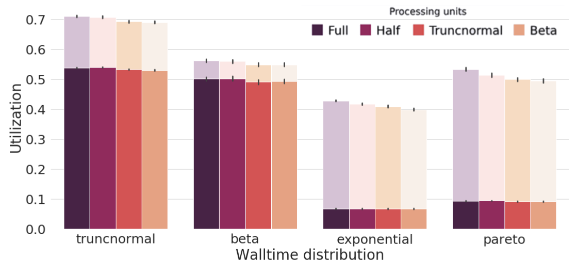
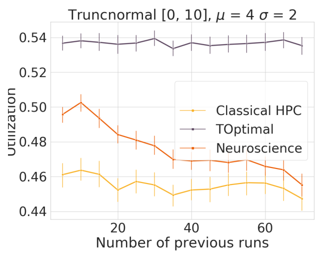


Figure 3: System utilization for different walltime distributions and processor allocations for the jobs.

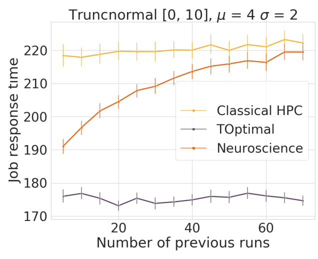
to the best result for the HPC scenarios. For Pareto and Exponential distributions, the improvements are over 1.5x and 2.5x respectively for the two metrics.

To better understand the impact of the processor allocations of the applications on the results, we simulate the same scenarios (same number of applications, following the same walltime distributions) when varying the processor allocation distributions for the jobs. Figure 3 shows the system utilizations under different distributions for the classical HPC approach (in the foreground) and our TOptimal algorithm (in the background). The results show that regardless of the jobs' walltime distributions and processor allocation distributions, using speculative scheduling can lead to benefit in system utilization (the same benefit was also observed for average job response time).

In the experiments, we observe that the Neuroscience approach gives better results than the classical HPC approach. Figure 4 shows

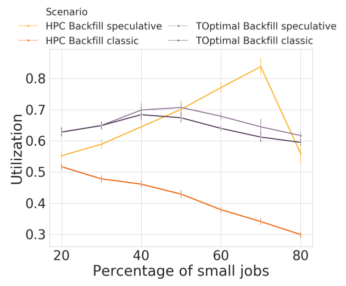


(a) Utilization

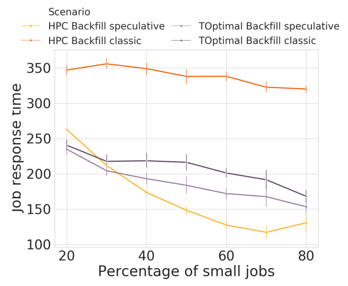


(b) Average job response time

Figure 4: Results when varying the number of previous runs used by the Neuroscience approach to compute the requests.



(a) Utilization



(b) Average job response time

Figure 5: Results when varying the percentage of smaller jobs within the total number of jobs in the simulation.

the utilization and job response time of TOptimal, classical HPC and Neuroscience when varying the number of previous runs used to compute the next request time by Neuroscience. Since each run generates random applications, TOptimal and HPC show slightly different results, within the margin of error, for each simulation. As expected, when using a longer history for computing the requests, Neuroscience's results approach those of the classical HPC approach (which is equivalent to using all past runs of an application). The best results seem to be when using approximately 10 previous runs. Thus, for the remaining simulations in this paper, we will use Neuroscience with 10 previous runs.

5.4 Speculative Backfilling (Scenario 2)

The simulations presented in the previous section contain mostly large jobs that are not particularly useful for backfilling algorithms. In this section, we relax this constraint and allow smaller jobs to run together with the large ones. We analyze the impact of speculative backfilling on the performance of the scheduling algorithms.

Figure 5 shows the system utilization and average job response time when varying the percentage of smaller jobs within the total number of jobs, which is fixed to be $M = 200$. The execution times of all jobs follow the Truncated Normal distribution. The large ones are between 1 hour and 8 hours with $\mu = 4$ and $\sigma = 2$, and the small ones are between a few seconds and 4 hours with $\mu = 1$ and $\sigma = 1$. Compared to the classic backfilling approach, speculative backfilling drastically improves the performance for the HPC scheduler while giving only slight improvements for the TOptimal scheduler. This is due to the use of speculative scheduling in TOptimal, which produces more restricted walltime requests, thus not leaving much room for further backfilling. In this case, once the percentage of small jobs exceeds 50%, the HPC scheduler does not need speculation at submission time and can achieve good performance with just speculative backfilling.

5.5 Considering Backfilling Jobs (Scenario 3)

The reservation sequence given by TOptimal does not take into consideration the availability of small backfilling jobs in the system. In this section, we focus on understanding the limitations of TOptimal and on analyzing the benefits of ATOptimal that considers the additional amount of work introduced through backfilling jobs, which can vary on an HPC system depending on the day. We use the sequence generated by ATOpimal to overwrite a large job's requests in the presence of these backfilling jobs, and compare the results with those of TOptimal and the HPC scheduler.

In this experiment, the execution times of all large jobs follow the Truncated Normal distribution between 1 and 20 hours with $\mu = 8$, $\sigma = 2$, while the system receives an incoming stream of backfilling jobs, whose execution times follow the same distribution but scaled down by a factor of 100. These jobs are uniformly distributed throughout the simulation time window to simulate a continuous influx of available work with an arrival rate λ, which is configured such that the normalized work rate Z/P is varied between 0.1 and 0.9. We assume these backfilling jobs are predictable and we use the exact execution times as their requested times.

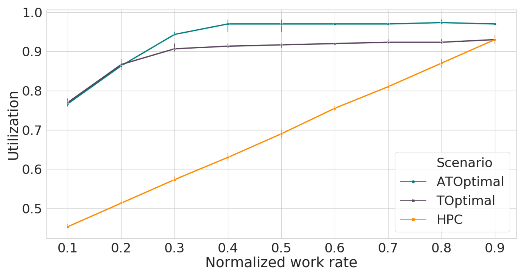
Figure 6 presents the system utilization and average job response time when we vary the normalized work rate for backfilling jobs. As expected, the ATOptimal sequence gives results that are close to those of TOptimal for small work rates and approach those of the HPC scheduler for larger rates. Overall, the system utilization increases with the work rate and is always higher for ATOptimal. For average job response time, ATOptimal falls in between the other two, and as we increase the work rate, backfilling jobs start to experience longer waiting times, thus driving up the average response time of all jobs in the system.

Figure 7 shows the average response time when looking only at the large jobs. The HPC scheduler's large requested times leads to the highest response time, while the conservative requested times of TOptimal gives the lowest response time. The ATOptimal algorithm again falls in between the two, with a response time that increases with the work rate, due to the use of increasingly larger requested times for the large jobs.

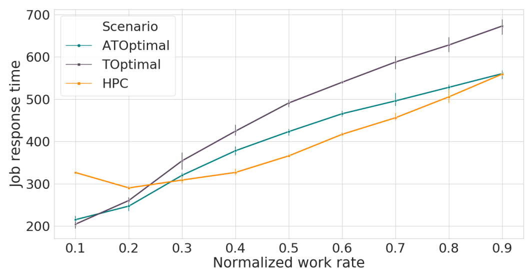
5.6 Two Weeks in an HPC System's Life (Scenario 4)

In this section, we simulate the execution on a large-scale HPC system (Intrepid) with realistic stochastic applications.

Intrepid was a BlueGene/P supercomputer at the Argonne National Laboratory between 2008 and 2014 (ranked 3rd on the June 2008 Top500 list). We use the logs from [19] to analyze the application submissions on Intrepid [2] between January 5 to September 1, 2009. Table 2 presents the main characteristics of the machine



(a) Utilization



(b) Average job response time

Figure 6: System utilization and average job response time when varying the normalized work rate for backfilling jobs.

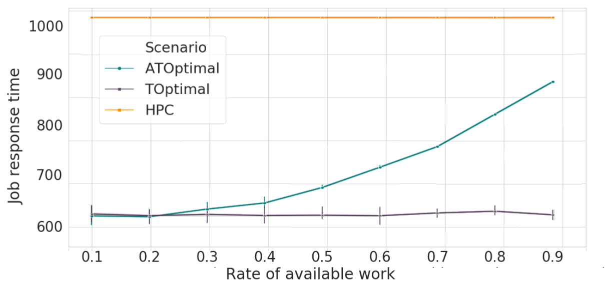
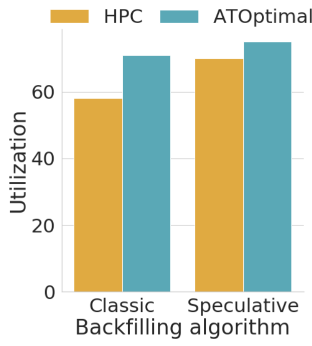


Figure 7: Average response time only for large jobs when varying the normalized work rate for backfilling jobs.

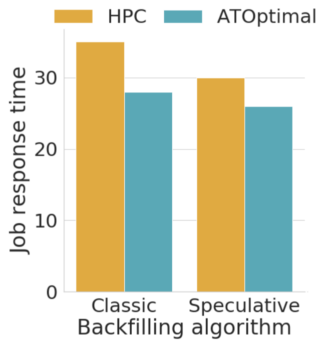
Table 2: Characteristics of the Intrepid system.

Total submissions	68936
Over-estimated submissions	82.2 %
Under-estimated submissions	17.7%
Average backfilling time	1.36 hours
Average backfilling space	2132 node hours
Average job size	880 nodes / 3089 node hours
Average small jobs size	48.6 nodes / 31 node hours
Percentage of small jobs	30.8%
Unused backfilling space	2.8 hours/day
Normalized rate of backfilling work	0.21

during this period. Specifically, over 82% of the submissions requested more time than the actual walltime of the job, averaging 1.36 hours of backfilling time. While the majority of these jobs were over-estimated by a maximum of 4 hours, there are cases where the over-estimation created more than 10 hours of backfilling time. The volume of these backfilling gaps (processing units times backfilling time) is over 2000 node hours on average. The number of small jobs running on Intrepid represents a third of all applications, but they only account for <1% of the total work in the system. Even if all the small jobs were used by the backfilling algorithm, the gap created by over-estimation during the analyzed timeframe would keep the system idle for almost 3 hours/day on average.



(a) Utilization



(b) Average job response time

Figure 8: Results for simulating two weeks of neuroscience applications' execution on Intrepid.

A wide range of scientific and engineering applications were run on Intrepid, including some large-scale simulation codes for chemistry, astrophysics, genetics and material science. The logs do not provide any information on the actual applications ran during the specified timeframe. For our simulations, we instead use the neuroscience applications from the Vanderbilt medical imaging database [12]. We chose three applications whose characteristics are presented in Table 3. We simulate 2 weeks of execution for these applications on Intrepid with a total of 400 submissions, while matching the normalized work rate of 0.21 for backfilling jobs (simulated by the FSL application) seen on Intrepid (Table 2).

The results of the simulation are presented in Figure 8. We can see that ATOptimal improves both system utilization and average job response time by about 20% compared to HPC using classic backfilling. Additionally, speculative backfilling further improves the performance for both HPC and ATOptimal. Overall, our speculative techniques provide 25-30% improvement over the classical HPC scheduling approach.

6 CONCLUSION AND FUTURE WORK

There is currently an increasing number of stochastic and unpredictable applications that use HPC platforms. In this paper, we

Table 3: Characteristics of three neuroscience applications used in the simulation.

Neuroscience Application	Walltime Distribution	# Submissions
Abdominal multi-organ segmentation code (AbOrganSeq_Whole_v1) [23]	Truncated Normal from 11 to 31 hours; $\mu = 20$ and $\sigma = 8$	10
A whole brain segmentation and cortical reconstruction (maCRUISE) [14]	Truncated Normal from 1.5 to 3 hours; $\mu = 1.7$ and $\sigma = 0.5$	90
FSL library [26] of analysis tools for fMRI, MRI and DTI brain imaging data	Truncated Normal from 10 to 35 minutes; $\mu = 20$ and $\sigma = 8$	300

have shown that speculative scheduling has a great potential for improving both system-level and user-level performance for these applications. Our solutions can be easily integrated into the current HPC schedulers by configuring the runtime system to use past statistics of the applications upon submissions. This allows the schedulers to compute speculatively the request reservation and backfilling decisions for the jobs. Experimental results show that speculative scheduling can improve system utilization and average job response time by 25-30%. While this paper focuses on request time, our model can be extended to other resources as well.

We seek to investigate multiple directions in the future. Adding other resources in the model is a priority, as well as analyzing characteristics of current HPC systems and including more complex distributions. Another direction is to create more disruptive scheduling solutions that would not kill a job that exceeds the allocated time but adapt other reservations to accommodate a longer run.

REFERENCES

[1] Guillaume Aupy, Ana Gainaru, Valentin Honoré, Padma Raghavan, Yves Robert, and Hongyang Sun. 2019. Reservation Strategies for Stochastic Jobs. In *IEEE International Parallel and Distributed Processing Symposium.*

[2] Abhinav Bhatele, Lukasz Wesolowski, Eric Bohm, Edgar Solomonik, and Laxmikant V. Kale. 2010. Understanding Application Performance via Micro-benchmarks on Three Large Supercomputers: Intrepid, Ranger and Jaguar. *The International Journal of High Performance Computing Applications* 24, 4 (2010), 411–427.

[3] Louis-Claude Canon, Aurélie Kong Win Chang, Yves Robert, and Frédéric Vivien. 2018. *Scheduling independent stochastic tasks under deadline and budget constraints.* Research Report 9178. INRIA.

[4] Louis-Claude Canon and Emmanuel Jeannot. 2010. Evaluation and optimization of the robustness of dag schedules in heterogeneous environments. *IEEE Transactions on Parallel and Distributed Systems* 21, 4 (2010), 532–546.

[5] N. Capit, G. Da Costa, Y. Georgiou, G. Huard, C. Martin, G. Mounie, P. Neyron, and O. Richard. 2005. A batch scheduler with high level components. In *CCGrid.* 776–783.

[6] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM* 51, 1 (Jan. 2008), 107–113.

[7] F. Dong, J. Luo, A. Song, and J. Jin. 2010. Resource Load Based Stochastic DAGs Scheduling Mechanism for Grid Environment. In *IEEE 12th International Conference on High Performance Computing and Communications (HPCC).* 197–204.

[8] Ana Gainaru et al. 2019. ScheduleFlow: A simulator for HPC schedulers. https://github.com/anagainaru/SchedulerSimulator. [Online; accessed 19-April-2019].

[9] Dror G. Feitelson, Larry Rudolph, and Uwe Schwiegelshohn. 2005. Parallel Job Scheduling — a Status Report. In *Proceedings of the 10th International Conference on Job Scheduling Strategies for Parallel Processing.* 1–16.

[10] Ana Gainaru, Hongyang Sun, Guillaume Aupy, Yuankai Huo, Bennett A Landman, and Padma Raghavan. 2019. On-the-fly scheduling versus reservation-based scheduling for unpredictable workflows. *The International Journal of High Performance Computing Applications* (2019).

[11] Bruno Gaujal and Jean-Marc Vincent. 2009. Comparisons of Stochastic Task-Resource Systems. In *Introduction to Scheduling*. Springer, Chapter 10.

[12] Robert L. Harrigan, Benjamin C. Yvernault, Brian D. Boyd, Stephen M. Damon, Kyla David Gibney, Benjamin N. Conrad, Nicholas S. Phillips, Baxter P. Rogers, Yurui Gao, and Bennett A. Landman. 2016. Vanderbilt University Institute of Imaging Science Center for Computational Imaging XNAT: A multimodal data archive and processing environment. *NeuroImage* 124 (2016), 1097 – 1101.

[13] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. 2011. Mesos: A Platform for Fine-grained Resource Sharing in the Data Center. In *the 8th USENIX Conference on Networked Systems Design and Implementation.* 295–308.

[14] Yuankai Huo, Andrew J. Plassard, Aaron Carass, Susan M. Resnick, Dzung L. Pham, Jerry L. Prince, and Bennett A. Landman. 2016. Consistent cortical reconstruction and multi-atlas brain segmentation. *NeuroImage* 138 (2016), 197 – 210.

[15] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. 2007. Dryad: Distributed Data-parallel Programs from Sequential Building Blocks. In *the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems.* 59–72.

[16] K. Li, X. Tang, B. Veeravalli, and K. Li. 2015. Scheduling Precedence Constrained Stochastic Tasks on Heterogeneous Cluster Systems. *IEEE Trans. Comput.* 64, 1 (2015), 191–204. https://doi.org/10.1109/TC.2013.205

[17] Michael L. Pinedo. 2008. *Scheduling: Theory, Algorithms, and Systems* (3rd ed.). Springer.

[18] Garrick Staples. 2006. TORQUE Resource Manager. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing (SC '06).* Article 8.

[19] W. Tang, Z. Lan, N. Desai, D. Buettner, and Y. Yu. 2011. Reducing Fragmentation on Torus-Connected Supercomputers. In *2011 IEEE International Parallel Distributed Processing Symposium.* 828–839. https://doi.org/10.1109/IPDPS.2011.82

[20] Xiaoyong Tang, Kenli Li, Guiping Liao, Kui Fang, and Fan Wu. 2011. A Stochastic Scheduling Algorithm for Precedence Constrained Tasks on Grid. *Future Gener. Comput. Syst.* 27, 8 (2011), 1083–1091.

[21] H. Topcuoglu, S. Hariri, and Min-You Wu. 2002. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE TPDS* 13, 3 (March 2002), 260–274. https://doi.org/10.1109/71.993206

[22] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. 2013. Apache Hadoop YARN: Yet Another Resource Negotiator. In *the 4th Annual Symposium on Cloud Computing.* 5:1–5:16.

[23] Yan Wang, Yuyin Zhou, Wei Shen, Seyoun Park, Elliot K. Fishman, and Alan L. Yuille. 2018. Abdominal multi-organ segmentation with organ-attention networks and statistical fusion. *CoRR* abs/1804.08414 (2018).

[24] Ole Weidner, Malcolm Atkinson, Adam Barker, and Rosa Filgueira Vicente. 2016. Rethinking High Performance Computing Platforms: Challenges, Opportunities and Recommendations. In *Proceedings of the ACM International Workshop on Data-Intensive Distributed Computing.* 19–26.

[25] Gideon Weiss. 1992. Turnpike Optimality of Smith's Rule in Parallel Machines Stochastic Scheduling. *Math. Oper. Res.* 17, 2 (1992), 255–270.

[26] Mark W. Woolrich, Saad Jbabdi, Brian Patenaude, Michael Chappell, Salima Makni, Timothy Behrens, Christian Beckmann, Mark Jenkinson, and Stephen M. Smith. 2009. Bayesian analysis of neuroimaging data in FSL. *NeuroImage* 45, 1, Supplement 1 (2009), S173 – S186.

[27] Andy B. Yoo, Morris A. Jette, and Mark Grondona. 2003. SLURM: Simple Linux Utility for Resource Management. In *JSSPP.* 44–60.